AD-A090 831 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
MUTATION ANALYSIS AS A TOOL FOR SOFTWARE QUALITY ASSURANCE.(U)
OCT 80 R A DEMILLO DAAG29-80-C-0120
UNCLASSIFIED GIT-ICS-80/11 NL

School of

Information and Computer Science

GEORGIA INSTITUTE OF TECHNOLOGY

LEVEL II

(12)

GIT-ICS-80/11

MUTATION ANALYSIS AS A TOOL FOR SOFTWARE QUALITY ASSURANCE †

Richard A. DeMillo*

October, 1980

*School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

†Work supported in part by U.S. Army Research Office, Grant #DAAG29-80-C-0120 and by Office of Naval Research, Grant #N00014-79-C-0231.

MUTATION ANALYSIS AS A TOOL FOR SOFTWARE QUALITY ASSURANCE

Richard A. DeMillo

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia

Software quality assurance is a protocol involving two parties: the vendor and the customer. As shown in Figure 1 below, the vendor produces a software system which he submits to the customer for purchase. The relationship between the vendor and the customer is left unspecified in this model, but the intent should be clear; the following table gives three common interpretations.

Vendor	Customer		
	Chief Team Programmer	Marketing Manager	Purchaser
Programmer	System-Level Testing		
Chief Team Programmer		Quality Assurance	
Marketing Manager			Acceptance

With any of these interpretations in mind, the protocol is easy to follow; the vendor presents to an impartial evaluator his software and evidence E which purports to show that the software performs as advertised to the customer. The evidence should be objective and the evaluation should be reproducible by both the vendor and the customer. The evaluation of the evidence is then submitted along with the software and the evidence to the customer who then makes the (subjective) decision of whether or not to accept the software.

Mutation analysis [1,2,3,4] is an evaluation technique. I will concern myself only with the correctness of the software; therefore, the evaluation that is returned is an indication of how well the software has been tested by the vendor, and the evidence used in the evaluation is a set of test cases.

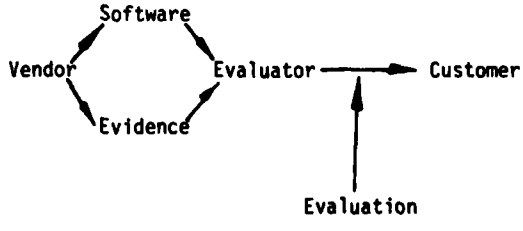


Figure 1.

The QA Protocol

The best evaluation which can be given to the customer is an objective probability that the program is correct, but there are compelling reasons for believing that such an approach is not feasible [5]. A good alternative approach is to let the evaluation represent a level of confidence in the adequacy of the test cases. Test cases are adequate if they demonstrate the correctness of the programs; in other words, in order to be adequate, the test cases must be so exhaustive that, not only must be proffered program run correctly on the test cases, but every incorrect program must run incorrectly. Unfortunately, this notion of adequacy is much too strong -- producing adequate test cases is impossible except for very simple programs. The fault is not with notion of adequacy; it is with the notion that test cases should be insensitive to other specialized information about the vendor. To be useful as evidence of a program's correctness, test data need only distinguish the program from finitely many alternatives -- the alternatives which correspond to the most likely errors to be introduced by the vendor. With this motivation, the evaluation resulting from a mutation analysis of a program P and its test data T (the mutation score, denoted ms(P,T)), can be precisely defined. A set of mutants of a program P, M(P), is a finite subset of the set of all programs written in the language of P. The set $EM(P) \subseteq M(P)$ consists of those programs in M(P) which are (functionally) equivalent to P. For a set of test cases T, DM(P,T) is the set of programs in M(P) which give results differing from P on at least one point in T. A mutation score for P,T is defined as follows:

$$ms(P,T) = \frac{|DM(P,T)|}{|M(P)| - |EM(P)|}$$

The central problem for quality assurance is then to choose a function M so that

1. ms(P,T) = 1 exactly when T demonstrates the correctness of P with a high level of confidence, and
2. ms(P,T) is a relatively cheap measure to compute.

The advantage of such a measure is that it satisfies the basic requirements of the QA protocol. Since the sets M(P) and EM(P) are fixed beforehand, the measure is objective and since DM(P,T) depends only on being able to execute P-like programs, the measure is reproducible. Condition 1 insures that the results of the evaluation are reliable, and Condition 2 provides that the evaluation process will not be excessively burdensome to apply.

In a series of automated mutation analysis systems, the concepts of choosing M(P) by making "simple" mutations has been explored (see, e.g., [3]). The underlying assumption for such a choice has been that these mutations correspond to the errors most likely to be made in producing P. How good is test data T (the evidence) such that ms(P,T) = 1? An approach to this problem has been formulated in a recent thesis at Georgia Tech [6].

One measure of how good the ms measure might be is how many "complex" mutants it leaves "unexplained". An experiment to investigate this effect for Cobol programs is discussed in [6]. The programs P1-P6 are representative Cobol programs in the 100-700 line range. To test the measure ms(Pi,T) one first derives test data Ti so that

ms(Pi,Ti) = 1. The question then becomes how many complex mutants give the same results as Pi on Ti. Any such complex mutants are said to be uncoupled. A key point to be settled in such an experiment is what is meant by "complex." The experimental results shown in Figure 2 use complex mutants resulting from random pairs of simple mutants, while the results shown in Figure 3 use complex mutants resulting from random correlated pairs of simple mutants. A detailed justification for using random pairs and correlated pairs in two separate experiments is given in [3]. The quantity "survives" denotes the number of complex mutants that are left uncoupled by Ti, and "not equivalent" denotes the number of uncoupled mutants that are not functionally equivalent to Pi -- an important quantity since the functionally equivalent mutants cannot be distinguished by any test data and therefore do not effect the strength of the ms(Pi,Ti) measure. The diagrams in Figures 2 and 3 show the 95% confidence intervals on the quantity (z×100,000), where z is the probability that a randomly selected pair of simple mutants (correlated mutants) is uncoupled for test data Ti. The size of the samples (50,000 for Figure 1 and 10,000 for Figure 2) reflect the relative sparseness of the set of correlated simple mutants.

Program	Survives	Not Equiv.	95% c.i.
P1	26	0	0 -- 7.4
P2	12	0	0 -- 7.4
P3	22	5	3.2 -- 23.3
P4	10	2	0.5 -- 14.4
P5	45	0	0 -- 7.4
P6	13	0	0 -- 7.4

Figure 2.
50,000 Random Pairs of Mutants for each Pi

Program	Survives	Not Equiv.	95% c.i.
P1	0	0	0 -- 36.9
P2	3	1	0.3 -- 55.7
P3	60	19	114.4 -- 296.6
P4	3	3	6.1 -- 87.6
P5	1	0	0 -- 36.9
P6	1	0	0 -- 36.9

Figure 3.
10,000 Correlated Pairs for Each Pi.

Essentially the same experiment has been performed on other categories of complex mutants with even more dramatic results: there are no uncoupled mutants! The rare uncoupled mutants in these experiments seem to fall into three classifications, all of them tied to the way in which loops and decisions are tested. Since the number of execution paths in a looping program can be infinite and in a loop-free program can be exponential in the number of program statements, there is no computationally feasible method of testing all program paths -- the uncoupled mutants in these experiments seem to be due to this fact. Surprisingly, the results of [6] also seem to indicate that the existence of uncoupled mutants is not related to the branching complexity of the program.

Condition 2 above asks that the calculation of ms(P,T) be efficient. As is discussed in [3], the complexity of the calculation -- when expressed as |M(P)| -- tends to be roughly quadratic in the size of P. Since there are several heuristics available for speeding up the calculation of ms(P,T) from M(P), this is quite acceptable for programs in the 100-5000 line range. In fact, medium scale programs in the range of 1000 lines have been tested on a fully loaded DEC-20 at 5-10 times coding rates for similar production codes.

For evaluations of very large monolithic programs, it may be necessary to sample a small fraction of the available mutants in M(P) and use the results of the sample to infer the strength of the test data. In an experiment also reported in [6], the programs P1-P6 were tested using the sampling strategy and the resulting tests were then evaluated using conventional mutation analysis. Approximately 10% of the mutants in M(P) were selected at random for elimination. The resulting test data T1,...,T6 was used to calculate ms(Pi,Ti), with the following results:

ms(T1,P1) = 1
ms(T2,P2) = 1
ms(T3,P3) = .99
ms(T4,P4) = .99
ms(P5,T5) = .99
ms(P6,T6) = .99.

This procedure is clearly valuable in reducing the cost of mutation analysis, and as these results demonstrate, sacrifices very little in the way of evaluation.

Basic investigations into the effectiveness of mutation analysis as a tool for quality assurance continue. As the recent experimental [4,6] and theoretical [4] results suggest, however, mutation analysis can be a very attractive tool for practical QA as well as for program testing.

References

1. R. DeMillo, F. Sayward and R. Lipton, "Hints on Test Data Selection: Help for the Practicing Programmer," Computer, April, 1978, pp. 34-41.

2. T. Budd, R. DeMillo, F. Sayward and R. Lipton, "The Design of a Prototype Mutation System for Program Testing," Proceedings 1978 NCC, pp. 623-627.

3. A. Acree, T. Budd, R. DeMillo, R. Lipton, F. Sayward, "Mutation Analysis," Georgia Institute of Technology, Report GIT/ICS/79/08, September, 1979.

4. T. Budd, R. DeMillo, R. Lipton, F. Sayward, "Theoretical and Empirical Studies on Using Program Mutation to Test the Functional Correctness of Programs," Proceedings 1980 ACM Symposium on Principles of Programming Languages, January, 1980, pp. 220-233.

5. R. DeMillo and F. Sayward, "Statistical Reliability Theory," in Software Metrics (F. Sayward, editor), MIT Press (to appear in early 1981).

6. A.T. Acree, "On Mutation," Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: GIT-ICS-80/11	2. GOVT ACCESSION NO.: AD-A090831
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Mutation Analysis as a Tool for Software Quality Assurance	5. TYPE OF REPORT & PERIOD COVERED: Interim Technical Report
	6. PERFORMING ORG. REPORT NUMBER: GIT-ICS-80/11
7. AUTHOR(s): Richard A. DeMillo	8. CONTRACT OR GRANT NUMBER(s): ARO Grant #DAAG29-80-C-0120, ONR Grant #N00014-79-C-0231
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia 30332	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: October 1980
	13. NUMBER OF PAGES: 4
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

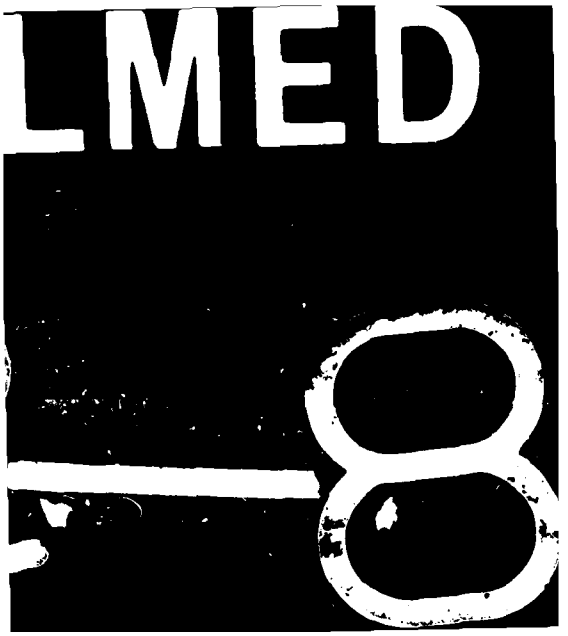
program mutation, program testing, quality assurance

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A protocol for using mutation analysis as a tool for software quality assurance is described. The results of experiments on the reliability of this method are also described.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

410044


LMED
8